LA-UR-19-20233
Approved for public release; distribution is unlimited.

Title: (U) New xRage Capabilities Enabled by Infrastructure Cleanup

Author(s): Ferenbaugh, Charles Roger
Ellinger, Carola
Gaspar, Andrew James
Hall, Michael L.
Krueger, Brendan K.
Masser, Thomas
Swaminarayan, Sriram

Intended for: meeting proceedings from NECDC 2018

Issued: 2019-01-14

(U) New xRage Capabilities Enabled by Infrastructure Cleanup

[1]**Charles Ferenbaugh,** [1]**Carola Ellinger,** [1]**Andrew Gaspar,** [2]**Michael L. Hall,**
[3]**Brendan Krueger,** [2]**Thomas Masser,** [1]**Sriram Swaminarayan**
[1]*CCS-7 Applied Computer Science, LANL*
[2]*CCS-2 Computational Physics and Methods, LANL*
[3]*XCP-2 Eulerian Codes, LANL*

cferenba@lanl.gov, 505-665-7045, MS T087
October 16, 2018

Abstract

For the past several years, the Eulerian Applications Project at LANL has been working to modernize the xRage code base. Initial efforts, done in FY15-16 and presented in previous work, focused primarily on splitting the code into meaningful packages and levelizing the dependency structure [1, 2]. Ongoing efforts include further refactoring across packages, code cleanup within packages, and writing of unit tests (Fig. 1).

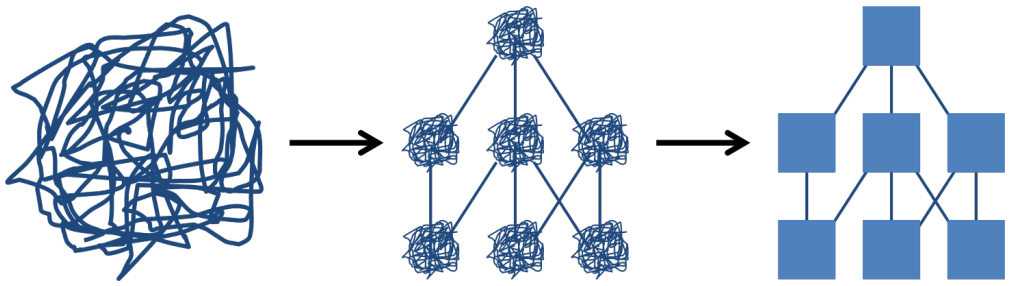


Figure 1: Notional diagram of xRage code refactoring, phases 1 and 2.

When we started the modernization process, one of our major goals was to simplify the code base and enable future code changes, in support of advanced architectures and also of new physics methods. As a result, several recent xRage improvements have been enabled by this infrastructure cleanup. This paper describes some of these improvements, and the infrastructure cleanup tasks that enabled them.

C++ Conversion of the Token Module

We have deployed a C++ implementation of the xRage "token" module, a communication utility that can memoize and reuse MPI point-to-point communication patterns such as cross-processor neighbor lookups. This improves over the existing Fortran version by using C++ templates and Kokkos views, allowing for improvements such as reduction of repeated code and processing of non-contiguous multidimensional Fortran arrays without data copies. Also, initial testing suggests that the C++ version is slightly faster than the Fortran version. (A separate NECDC 2018 paper by Andrew Gaspar describes this work in more detail.)

The writing of the C++ version was made much easier by the encapsulation work previously done on the Comm package. Because of this, the token module had a well-defined, relatively simple API; and there were no implementation constraints on the C++ version, so long as it conformed to the same API. It was also useful that the Fortran version was exercised by a unit test suite, and a C++ unit test harness was in place that allowed these tests to be converted and used to verify the C++ version.

Performance Portability Experiments

We are in the process of writing prototype performance-portable versions of two key xRage kernels (derivative calculation and CG solver), introducing threading by using OpenMP directives or by using the Kokkos framework. Since Kokkos is a C++-specific framework, this effort required reimplementing the relevant kernels in C++ and introducing interoperability machinery to call the C++ kernels from within the larger Fortran code.

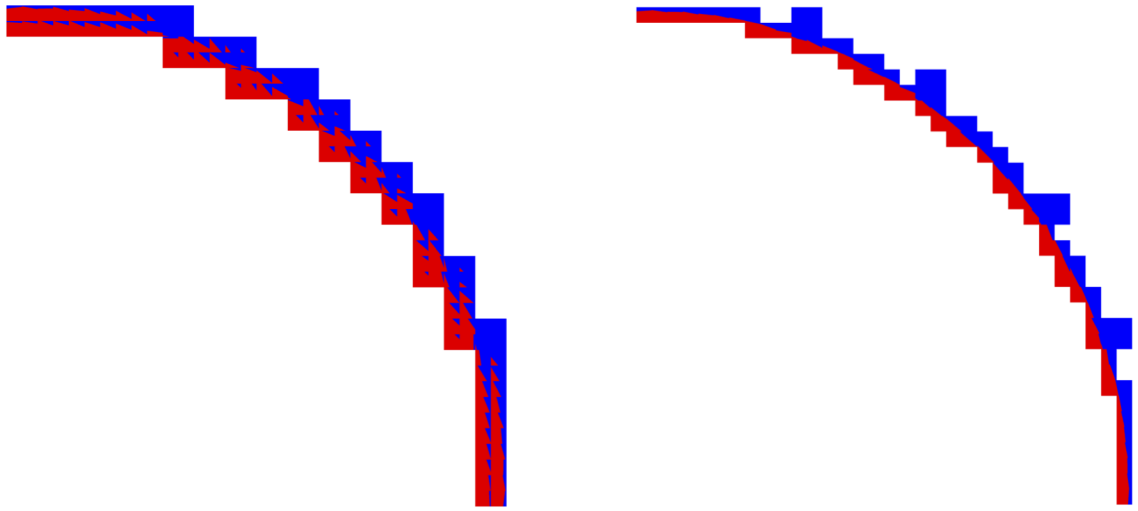
Previous work had already been done to clean up the derivative and CG kernels, encapsulating them and removing unnecessary dependencies on other parts of the code. This allowed us to optimize the kernels and write the C++ versions in isolation. This also allowed us to leverage the existing unit test machinery and verify correctness of the optimized kernels before testing performance. (A separate NECDC 2018 paper by Geoff Womeldorff describes this work in more detail.)

(a) (b)

Figure 2: Target platforms for performance optimization: (a) Trinity and (b) Sierra.

Interface-aware AMR

Recently it was discovered that the xRage AMR and variable reconstruction process ("recon") did not account for material interfaces when mapping physics variables into newly refined cells (see for example Fig. 3a). To fix this, it was necessary to add calls from recon into various material interface routines. Unfortunately this wasn't possible in the dependency hierarchy that existed at the time; the recon code was part of the Mesh package, and MaterialInterface already depended on Mesh, so the new calls would have created a circular dependency (Fig. 4a).



(a) (b)

Figure 3: Material interfaces in refined cells (a) before and (b) after Recon code fixes. Notice the red and blue slivers in (a).

To enable the new calls from recon, the Mesh package was refactored so that Recon could be removed and made into a separate package (Fig. 4b). This refactor leveraged work that had been done earlier, to make mesh generation available to unit tests; only a moderate amount of additional work was needed to separate Mesh from Recon. Also, the unit tests for Mesh and several physics packages were dependent on Mesh but not on the recon functionality. Because of this, we could verify that the Mesh/Recon refactor had been done properly by checking that these unit tests continued to build and pass after the refactor.

Portage Remapping

We are adding an interface from xRage to the Portage remap library, developed by LANL's Ristra next-generation code project. This interface will initially be used for remapping from the xRage AMR mesh to the Generalized Eulerian Mesh (GEM) type used by some third-party libraries. Development of this interface posed several challenges. While xRage is almost entirely written in Fortran, Portage is written in C++ and makes significant use of C++ features such as templated classes and functors. Also, the existing

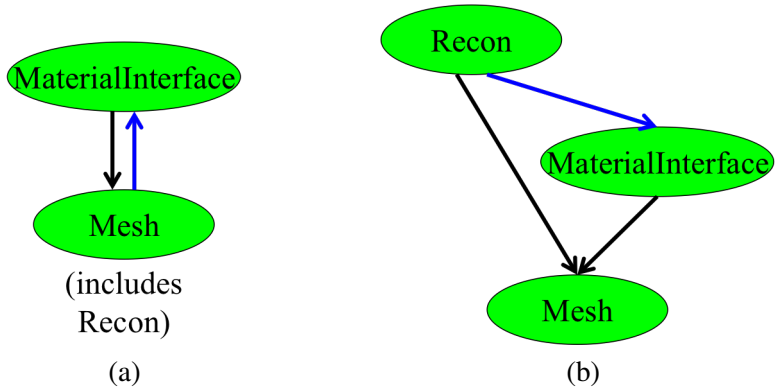


Figure 4: Dependencies between Mesh, MaterialInterface, and Recon (a) before refactor, and (b) after refactor. Blue arrows represent new functionality to be added for interface-aware AMR.

GEM remapper had historically been coded in such a way that the distinction between the GEM data type and the mapper was not clear, making it difficult to write new code using the data type but not the mapper.

To enable the new interface, the needed C++ routines from Portage were wrapped by Fortran-callable wrapper routines, taking advantage of the Fortran/C interoperability features in Fortran 2003. Mesh data structures from Fortran were wrapped in C++ classes, allowing Portage to use the xRage data without having to make copies of problem-sized arrays. All of this was made possible by cleanup work that had been done recently in the Interpolators package in xRage, first to encapsulate the package from the rest of the code, and then to clean up data structures within the package. We were also able to leverage the unit tests that had been written for the existing mapper, and adapt them to verify that the Portage mapper was working correctly (see Fig. 5).

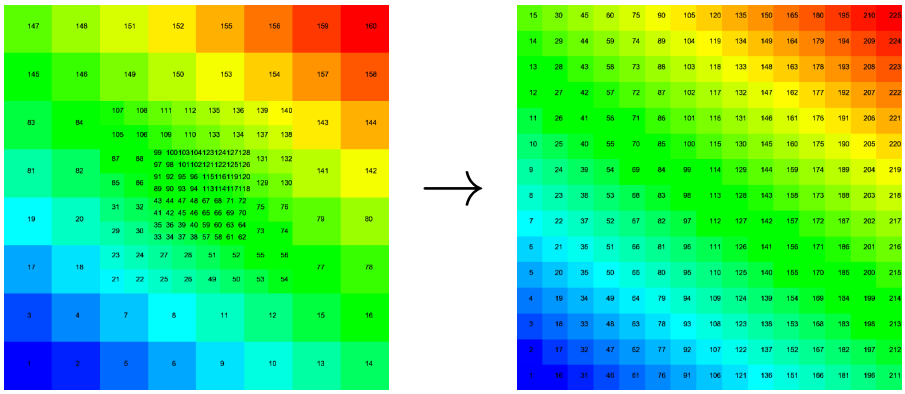


Figure 5: Portage remap from an xRage AMR mesh to a Generalized Eulerian mesh (GEM) for third-party library use.

Unsplit Hydro Method

xRage has historically used a directionally-split hydro method, i.e., x-, y-, and z-dimensions are updated separately. A new unsplit hydro method, expected to give more accurate results, is under development.

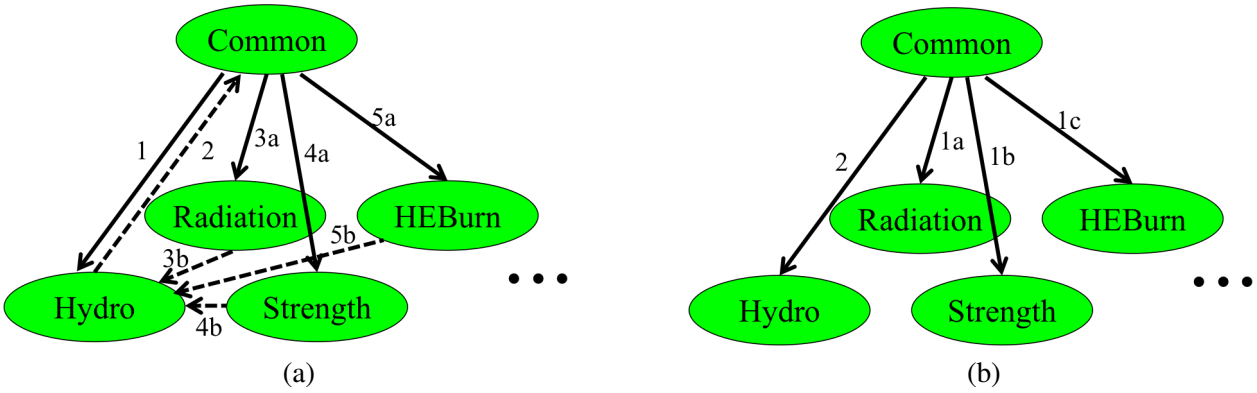


Figure 6: Dependencies between Hydro, Common, and other physics packages (a) with current advection approach, and (b) with anonymous advection. Dotted arrows represent callbacks.

As part of the hydro update, xRage needs to advect all of the state variables owned by the various physics packages. This is done by using an appropriate hydro routine as a callback, which is sent to each of the packages in turn and applied to that package's state variables (Fig. 6a). Certain assumptions are inherent in the current approach: it assumes that the advection can be done in a single step, and that the update can go directly from the initial to the final state, with no intermediate state needed. Unfortunately, the unsplit hydro method doesn't meet these assumptions; it may compute advection updates in multiple stages, with storage of intermediate state needed between stages. So the new hydro algorithm would require either a more complex callback interface, or an entirely different approach to handling advection.

To support the new hydro method, we are developing a new treatment of multiphysics advection which we call *anonymous advection* (see Fig. 6b). In this approach, a "dictionary" of advecting state variables is gathered from the physics packages before hydro starts. The dictionary contains pointers to variables, and a small amount of metadata for each variable (advect by volume vs. mass, extensive vs. intensive, etc.). The dictionary is sent as an input to the hydro update, which can then iterate through the dictionary and advect the variables in it, without having to call the physics packages or know which package provided which variable.

When anonymous advection is completely deployed, it will enable the full implementation of unsplit hydro, while also allowing the advection machinery in the existing hydro methods to be greatly simplified.

Acknowledgements

We wish to acknowledge the following contributors:

- Geoff Womeldorff, Robert Pavel, Joshua Payne (all CCS-7) and Bobby Philip (CCS-2) are developing the OpenMP and Kokkos kernel implementations.
- William Dai (XCP-2) implemented the interface-aware AMR approach and provided the pictures.
- Josh Dolence (CCS-2) is developing the unsplit hydro method.
- Chad Meyer (XCP-4) helped refine the initial concept of anonymous advection.

References

[1] Ferenbaugh, C., S. Swaminarayan, C. Aldrich, M. Calef, J. Campbell, *et al.*, EAP Code Modernization, Part I: Packagization. In *Proceedings of the 19th Nuclear Explosives Code Developers Conference*, Livermore, CA, October 17–21, 2016 (2017).

[2] Ferenbaugh, C., S. Swaminarayan, C. Aldrich, M. Calef, J. Campbell, *et al.*, Modernizing a Long-Lived Production Physics Code. Poster presented at Supercomputing 2016, Salt Lake City, UT, November 13–18, 2016.

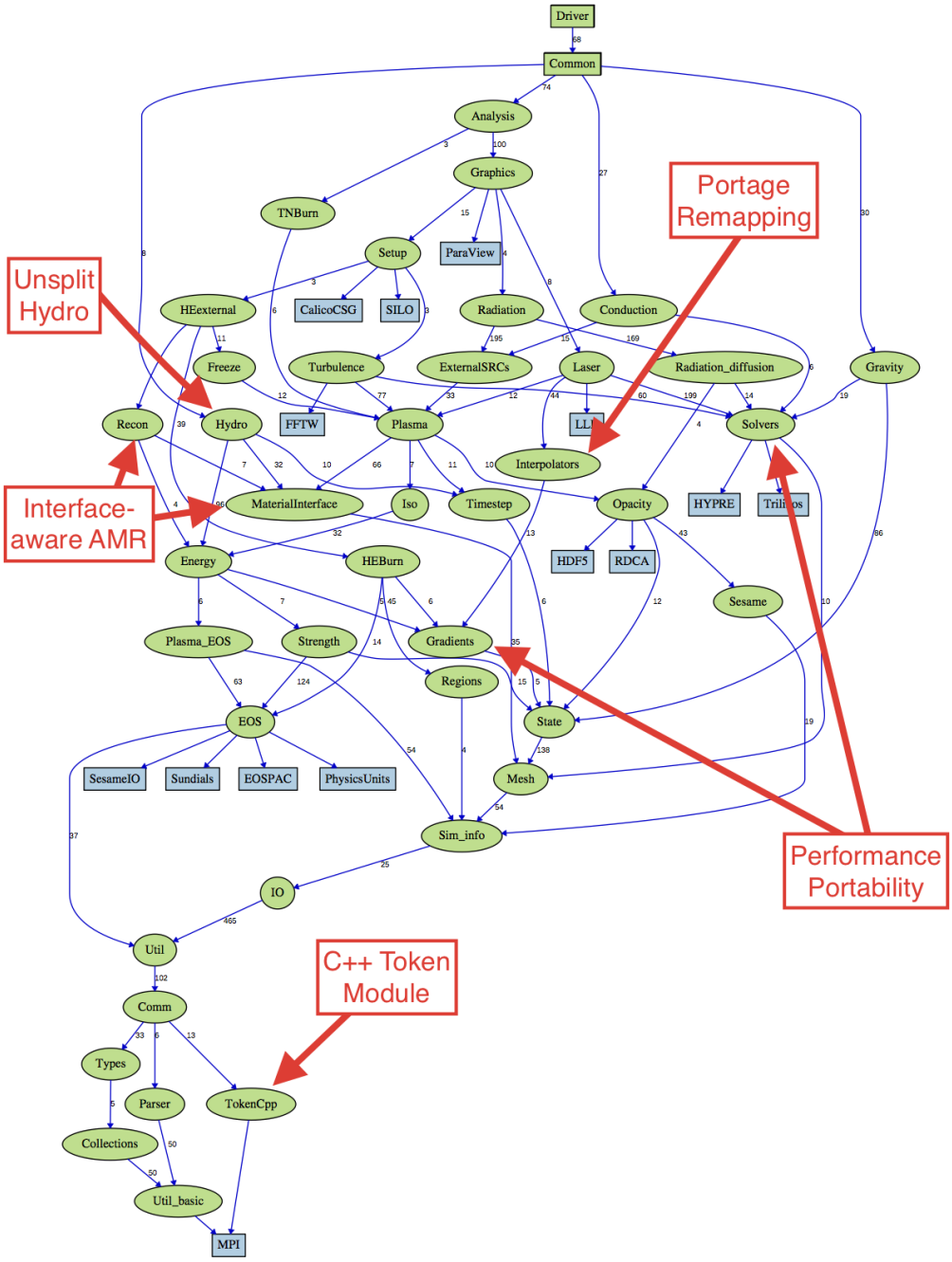


Figure 7: xRage package hierarchy. Packages associated with the work in this paper are called out by red boxes and arrows.